MSX

Memory Manager

Uebersetzung by A. Küpfer

005emMan, der MSX Memory Manager

Anfangs 1990 rief MSX Computer Magazin fürs erste die Besten MSX Programmierer von Holland zusammen um dem MSX Markt wieder etwas Power einzuhauchen. Die Programmierer lernten einander kennen und tauschten Ideen aus. Es schien eine Nachfrage nach einem Memory Manager vorhanden zu sein. Ein Programm, welches die Kapazitäten des MSX's beherrschen würde.

Mit dem Memory Manager werden zwei Teile angestrebt:

1) Das Suchen und Benutzen von Kapazitäten wird einfacher.

2) Es wird möglich mehrere verschiedene Programme zur gleichen Zeit in den Speicher zu laden, ohne dass sie einander im Wege stehen. Hierbei wird an RAMdisks, Printerbuffers und Freekick-ähnliche Programme gedacht.

Mit Version 1.0 von MemMan - gezeigt am 9. September 1990 - ist die erste Grundlage erreicht. An der zweiten Grundlage wird nachwievor gearbeitet.

Um MemMan interessanter zu machen wurde unter MCM eine MemMan Anpassung programmiert: BK, ein Bestandskopierprogramm. Dieses Programm funktioniert nur, wenn vorab MemMan geladen wurde. Mit hilfe des Managers gebraucht BK dann beinahe alle Kapazitäten welche zu finden sind.
Nebst BK ist auch MTYPE dabei, eine Version von TYPE, welches den ganzen Bestand ladet, bevor es abgedruckt wird.

MemMan wird als Public Domain in die ganze Welt verschickt. Das heisst jeder kann von MemMan frei gebrauch machen. Es ist sogar erlaubt MemMan als einen Unterteil eines Softwarepaketes zu verkaufen. Nur so kann das Programm wachsen.

An MemMan haben mitgearbeitet:

Ramon van der Winkel
Robbert Wethmar
Ries Vriend
Paul te Bokkel
Markus The
Und eine Anzahl anderer, welche durch ihre Kritik MemMan zu dem gemacht haben, was es heute ist.

Die Prinzipien

MemMan verteilt die anwesenden Kapazitäten in Segmente von je 16 KB. Bevor ein Segment gebraucht werden kann, muss es angefragt werden. Nach Gebrauch muss man es wieder freigeben. Es gibt zwei Sorten von Segmenten: Die sogenannte pagina-spezifische genannt PSEG's oder die flexiblen genannt FSEG's.
PSEG's sind Segmente die auf einer bestimmten pagina angefragt werden, z.B. von &h4000-&h7FFF oder von &h8000-&hBFFF. Wenn ein PSEG angefragt wird, wird MemMan so schnell wie nur möglich die Kapazitätssegmente zuweisen, welche nicht im Memory Mapper sind. FSEG's sind Segmente die auf irgend einer pagina eingeschaltet werden können. Diese Segmente kommen immer aus Memory Mappers. Welche Sorte auch angefragt wird, MemMan sendet ein 16-Bit 'Segmentcode' zurück. Dieser Segmentcode wird wieder benötigt um einzuschalten und freizugeben. Wer aber nur Kapazitäten braucht zwischen &h8000 bis &hBFFF, fragt am besten PSEG's an. MemMan benutzt dann erst soviel wie möglich Kapazität aus den 'alten' 16 und 64 KB Modulen und dann erst den Memory Mapper. Mit Hilfe des MemMan's muss in diesem Falle nie mehr nach freien Speicherkapazitäten gesucht werden. Simpel eine pagina anfragen, benützen und schlussendlich wieder freigeben. So einfach ist das. Uebrigens gibt es eine pagina, welche sich mit MemMan nicht schalten lässt. Pagina 3 beinhaltet den MemMan code selbst und auch den stack (meistens) und eine grosse Menge Systemvariabeln. Es gibt auch noch einige Haken beim wegschalten des Ganzen.

Das Installieren

MemMan 1.0 gibt es in zwei Versionen: ein .BIN und ein .COM File. Es ist wohl klar, dass die .BIN Version aus dem BASIC gestartet werden muss, und die .COM Version im MSX-DOS System. Beide Versionen kehren, nach dem sogenannten warm boot, wieder ins BASIC zurück. Mit Hilfe des CFGMMAN.BAS ist es möglich MemMan nach dem Laden etwas im Tastenboardbuffer zu setzen. Auf diese Weise kann z.B. direkt ins MSX-DOS zurückgekehrt werden, oder ein anderes Programm aufgerufen werden. MemMan Version 1.0 lässt nach der Installation ungefähr 675 Bytes Besteuerungsprogramm, und dies wird mit jedem anwesenden RAM-Segment um je 3 Bytes erhöht. Platz für extra ROM Segmente sind schon vorgesehen. Wenn MSXDOS 2 benutzt wird, wird MemMan gemäss den Zeilen alle verfügbaren Kapazitäten einsammeln. MemMan arbeitet also jederzeit. Der grosse Vorteil von MemMan liegt darin, dass es auch alte Applikationen und Erweiterungen akzeptiert. Darum arbeitet MemMan auch, wenn MSXDOS2 nicht anwesend ist. Bevor sich MemMan ins RAM speichert, schaut es erst, ob schon eine Version vorhanden ist. Ist dies der Fall, erscheint eine Meldung am Bildschirm, die mitteilt, dass die Version X.X schon installiert wurde. Weiter passiert nichts.

Das aufrufen

Alle MemMan Routinen können aufgerufen werden via einem Hook, nämlich EXTBIO auf Adresse &hFFCA. Dieser Hook ist speziel für BIOS Applikationen gemeint. Das Kanji-ROM und das Philips RS-232 Interface machen unter MSXDOS2 schon gebrauch von den Kapazitätsroutinen.
Indem man im D Register ein Identifikationscode setzt des gewünschten Programmes und im E Register eine Funktionsnummer, kann die entsprechende Routine aktiviert werden. Für MemMan ist der ID-Code &h4D, der ASCII-Code des Grossbuchstaben M. Ein anruf einer MemMan Routine besteht dann immer aus minimal:

```
LD D,&h4D
LD E,<Funktionsnummer>
CALL &hFFCA
```

Meistens wird noch eventuel extra initialisierung nötig sein, und weil alle Register gelöscht werden müssen eine Anzahl PUSH und POP Instruktionen gegeben werden.
Uebrigens muss als erstes die Funktion IniChk aufgerufen werden um zu kontrollieren ob MemMan anwesend ist. Es spricht für sich selbst, dass wenn MemMan nicht geladen ist, auch keine MemMan Funktionen gebraucht werden können.

Die Funktionen

Nach dem Aufruf beinhalten die Register noch keine brauchbaren Werte. Nur die Register worin die Resultate zurück kommen, stehen brauchbare Werte. Auch wenn bei dieser Version ein Register nicht durch MemMan verändert wurde, dann muss damit gerechnet werden, dass eine noch folgende Version dies trotzdem kann.

Hierunter zuallererst eine Liste der Funktionen nach Nummern sortiert.

```
00 Use0       - Schalte ein Segment auf pagina 0 ein
01 Use1       - Schalte ein Segment auf pagina 1 ein
02 Use2       - Schalte ein Segment auf pagina 2 ein
10 Alloc      - Frage ein Segment an
11 SetRes     - Gib einem Segment den Reserved Status
20 DeAlloc    - Gib ein Segment zurück
21 ClrRes     - Lösche den Reserved Status eines Segmentes
30 IniChk     - Initialisierungs Check
31 Status     - Status Report
32 CurSeg     - Momentan eingeschaltetes Segment anfragen
40 StoSlt     - Momentan geschalteter Slotstand aufschlagen
41 ResSlt     - Aufgeschlagener Slotstand herstellen
50 Info       - Verschiedene Gegebenheiten anfragen
```

Funktion: IniChk
Nummer : 30
Eingabe : Register A mit einem willkürlichen Wert
Ausgabe : A - Eigegebenes Register A mit "M" (&h4D) erhöht
DE - Versionnummer (D=HIGH, E=LOW)
Wann : Jedes Programm muss diese Funktion als erstes aufrufen
Ablauf : Bei Register A wird der Wert "M" (&h4D) aufgezählt. Des weiteren wird ein 'UnCrash' ausgeführt. Dies beinhaltet, dass alle Angefragten Segmente freigegeben werden, gemeint sind die, welche als 'Reserved' markiert sind.
Des weiteren werden momentan angeschaltete Slots umgeschaltet zu einem Segment und Segmentcode welche durch CurSeg aufgeschalten werden. Wenn ein Segment noch nicht in der Tabelle vorhanden ist, wird daraus ein Neues gemacht.
Maximal können 10 Segmente neu gemacht werden. Weil das RAM als Segment aufgeführt ist in der Tabelle, werden nur die 'lose' angeschlossenen Slots und ROM's in der Tabelle zusätzlich verlangt. Im Register DE kommt die Versionsnummer von MemMan zurück. In Register D der hohe Teil (vor dem Punkt) und in Register E der tiefe Teil (nach dem Punkt). D und E können einen Wert von 0 bis 9 haben, nicht aber "0" to "9".

MemMan Initialisierungscheck. Ein Vorbild:

```
LD L,255-"M"        ;Mit "M" dabei beinhaltet L den
                     Wert -1
LD D,"M"            ;MemMan ID Code
LD E,30             ;Subfunktion: IniChk
CALL EXTBIO         ;Erstes mal "M" bei L zählen
INC L               ;ist L nun -1?
JR Z,...            ;Ja, MemMan ist anwesend.
```

Natürlich ist es sicherer, wenn der Vergleich mit zwei verschiedenen Werten probiert wird.

Funktion: Status
Nummer : 31
Eingabe : Nichts
Ausgabe : HL - Anzahl anwesende Segmente
BC - Anzahl freie Segmente
A - 'connected' Code
Wann : Jeder gewünschte Moment

Ablauf : Die Segmenten Tabelle wird durchlaufen und vor jedem anwesenden Segment wird Register HL um 1 erhöht. Die 10 extra Entries werden nicht gezählt. Wenn ein Entry noch frei ist, wird das BC Register um eins erhöht. In Register A kommt der 'Connected' Status wieder zurück. Momentan ist hier nur bit 0 in gebrauch. Wenn Bit 0 gesetzt ist, sind die Mapper Support Routinen von DOS 2.20 anwesend.
Wenn IniChk ein neues Segment macht, dann wird Register HL auch um eins erhöht.

Funktion: Alloc
Nummer : 10
Eingabe : B = Pagina Sorte Code (0, 1, 2, 3)
Ausgabe : HL - Segmentcode (0000 = kein Segment mehr brauchbar)
Wann : Bei Segmentabfrage
Ablauf : Die Segmententabelle wird nach einem noch freien Segment durchsucht.
Wurde ein freies Segment gefunden, dann wird geschaut, ob das Segment von der richtigen Sorte ist. Die Sortencode, welche im B-Register mitgegeben werden können, stehen unten aufgeschrieben. Entspricht ein Segment nicht dem gesuchten Typ, dann wird weiter gesucht. Wenn das Ende der Tabelle erreicht wurde, und es wurde noch kein gefragtes Segment gefunden, so geht die Routine zurück mit dem Wert 0000 im Register HL. Die Routine ruft sich selbst zweimal an: Einmal mit dem angegebenen Sortcode in B, das zweite mal mit dem Sortcode 3 in Register B. Der zweite Durchlauf geschieht nur, wenn beim ersten 0000 im Register HL erschienen ist. Wenn ein PSEG0000, PSEG4000 oder PSEG8000 beim ersten Durchlauf nicht frei war, dann wird beim zweiten auf die selbe Weise ein FSEG geopfert.

Sortencodes: 0 - PSEG0000
1 - PSEG4000
2 - PSEG8000
3 - FSEG

Wenn Bit 7 des Sortcodes gesetzt ist, werden Segmente die nicht einen expanded Slot bevorzugen über denen derjenigen, welche in einem expanded Slot sind getan. Von den Sortcodes sind nur Bit 0, 1 und 7 gültig, der Rest wird generiert.
Jedes angefragte Segment muss zur Beendigung des Programmes wieder freigegeben werden mit der Funktion 20 (DeAlloc)

Funktion: DeAlloc
Nummer : 20
Eingabe : HL - Segmentcode
Ausgabe : Nichts
Wann   : Bei Segmentrückgabe
Ablauf : Es wird zuerst kontrolliert, ob HL einen legalen Segmentcode hat, sollte dies nicht der Fall sein, so wird eine Eingabe generiert und die Routine kehrt zurück. Sonst wird das Segment wieder freigegeben.

Funktion: SetRes
Nummer : 11
Eingabe : HL - Segmentcode
Ausgabe : Nichts
Wann   : Um einem Segment den Reserved Status zu geben
Ablauf : Selbe Funktion wie DeAlloc, nur wird nun der Reserved Status gesetzt.
Der Reserved Status muss aufgehoben sein (Funktion 21, ClrRes) bevor das Segment mit Hilfe von DeAlloc freigegeben wird.

Funktion: ClrRes
Nummer : 21
Eingabe : HL - Segmentcode
Ausgabe : Nichts
Wann   : Um den Reserved Status eines Segmentes zu holen
Ablauf : Selbe Funktion wie DeAlloc, nur wird nun der Reserved Status geresettet.

Funktion: Use0
Nummer : 00
Eingabe : HL - Segmentcode
Ausgabe : A - Resultatcode (0, -1)
Wann   : Bei Segmenteinschaltung auf pagina 0
Ablauf : Erst wird geschaut ob das Segment, welches eingeschaltet werden muss, ein PSEG0000 oder ein FSEG ist. Wenn dem nicht so ist, dann kehrt die Routine zurück mit dem Resultatcode -1 in Register A.
Sonst wird der Segmentcode aufgeschaltet für CurSeg und auf pagina 0 eingeschaltet. Danach kehrt die Routine zurück mit dem Wert 00 im Register A. Der Segmentcode der im Register HL eingegeben werden muss ist das Resultat der Funktion Alloc oder CurSeg.

Funktion: Use1
Nummer : 01
Eingabe : HL - Segmentcode
Ausgabe : A - Resultatcode (0, -1)
Wann : Bei Segmenteinschaltung auf pagina 1
Ablauf : Diese Routine macht dasselbe wie Use0, nur wird nun kontrolliert ob PSEG4000 oder FSEG da ist, und das Segment auf pagina 1 wird angeschaltet. Für mehr Details siehe Use0.

Funktion: Use2
Nummer : 02
Eingabe : HL - Segmentcode
Ausgabe : A - Resultatcode (0, -1)
Wann : Bei Segmenteinschaltung auf pagina2
Ablauf : Diese Routine macht dasselbe wie Use0, nur wird diesmal PSEG8000 oder FSEG kontrolliert und Segment auf pagina 1 eingeschaltet. Für mehr Details siehe bei Use0

Funktion: GetCur
Nummer : 32
Eingabe : B - Pagina Code (0, 1, 2, 3)
Ausgabe : A - Segmenttype Code (PSEG=0, FSEG=-1)
HL- Segmentcode (0000 = Nicht zu bestimmen)
Wann : Beim Bestimmen des aktiven Segments auf pagina 0, 1, 2 oder 3
Ablauf : In einer internen Tabelle stehen Segmentcode, welche durch IniChk eingeschaltet wurden und durch Use0, Use1 und Use2 beim jedem Aufruf geupdated wurden.
CurSeg holt den aktuellen Segmentcode der betreffenden pagina aus der Tabelle und setzt diesen in Register HL. Dieser Segmentcode ist direkt brauchbar bei Use0, Use1 und Use2 (abhängig des Segmenttyps). In Register A kommt eine Type-Andeutung zurück. Wenn A den Wert 00 beinhaltet, dann ist das Segment in HL und PSEG, wenn A den Wert -1 beinhaltet, dann ist es ein FSEG. ROM Segmente (z.B. der BASIC ROM welche unter BASIC in pagina 0 und 1 steht) werden als PSEG angegeben.
Wenn der aktuelle Segmentcode nicht bestimmt werden kann (durch IniChk), dann kommt der Wert 0000 zurück in Register HL. Dies kommt nur vor, wenn ein unbekanntes ROM-Slot eingeschaltet ist und kein Kapazitätraum vorhanden ist um einen neuen entry in der Segmententabelle zu erfassen. Diese Situation wird sich auf dem MSX2(+) nicht ergeben.

Funktion: StoSlt
Nummer : 40
Eingabe : HL - Bufferkapazität pointer (9 Bytes)
Ausgabe : Nichts
Wann : Wenn der aktive Kapazitätsbestand aufgegeben werden muss.
Ablauf : Diese Routine berechnet für pagina 0, 1 und 2 die aktuelle Speicherkapazitäten aus, und nimmt diesen in der Bufferkapazität auf.

Funktion: RstSlt
Nummer : 41
Eingabe : HL - Bufferkapazität pointer (9 Bytes)
Ausgabe : Nichts
Wann : Wenn ein aufgeschalteter Kapazitätsbestand zurückgestellt werden muss.
Ablauf : Diese Routine schaltet anhand der Gegebenheiten eine in den Buffer geladene Kapazität zurück. Anderst als die anderen MemMan Funktionen, arbeiten RstSlt und StoSlt nicht mit Segmentcoden. Sie arbeiten mit einer sortenartigen Struktur.

Funktion: Info
Nummer : 50
Eingabe : B - Subfunktionnummer
Ausgabe : HL - Abhängig von der Funktion
Ablauf : Anhand der eingetippten Funktionsnummer im Register B wird in Register HL eine Adresse zurückgegeben.

Funktion:
0 - Fastuse adres Use0
1 - Fastuse adres Use1
2 - Fastuse adres Use2

Wer echt das Meiste heraus holen will, kann direkt zu diesen Adressen springen, um die gewünschte Use-Routine zu erreichen. Dadurch wird nicht allein der Teil von MemMan, welches die Funktionnummer interpretiert umgangen, sondern auch alle anderen Programme die am Hooks hängen könnten.